ADDIS ABABA UNIVERSITY

COLLEGE OF TECHNOLOGY AND BUILT ENVIRONMENT

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

Driver Assistance, Safety, and Vehicle Management App

Software Design Specification

PREPARED BY: -

Name	ID Number	Stream
Bethel Dereje	UGR/1397/14	Software
Elizabet Yonas	UGR/6912/14	AI
Heran Eshetu	UGR/5016/14	AI
Soliyana Kewani	UGR/6041/14	Software
Yordanos Melaku	UGR/8211/14	AI

ADVISORS: Mr. Michael Sheleme

Date: Dec 22, 2025

TABLE OF CONTENTS

List of Figures

Definitions, Acronyms, and Abbreviations

- AI - Artificial Intelligence
- APIs – Application Programming Interface
- SDS - Software Design Specification
- SRS - Software Requirement Specification
- UML - Unified Modeling Language

1. Introduction

1.1 Purpose

The purpose of this Software Design Specification (SDS) is to provide a detailed technical blueprint for the design and implementation of the Driver Assistance, Safety, and Vehicle Management System. This document translates the approved Software Requirements Specification (SRS) into a structured system design that defines the system architecture, subsystem interactions, data flow, and component responsibilities.

This SDS serves as a reference for developers, testers, and reviewers by describing how functional and non-functional requirements will be realized through specific design decisions. It establishes a clear foundation for implementation, ensures consistency across development activities, and supports maintainability, scalability, and reliability of the system.

1.2 General Overview

The Driver Assistance, Safety, and Vehicle Management System is a mobile-based software solution designed to support drivers in managing vehicle maintenance, accessing safety and educational resources, interacting with a driver community, locating nearby services, and receiving AI-assisted guidance. The system also includes a companion application for garage owners to manage service availability and appointment workflows.

From a design perspective, the system follows a client–server architecture consisting of:

- A Driver Mobile Application developed using Flutter
- A Garage Mobile Application developed using Flutter
- A Backend Server implemented using Node.js.

- A Centralized Database (PostgreSQL or MongoDB)
- Integration with external services such as Google Maps API and AI services.

The system is organized into modular subsystems, each responsible for a specific domain, including vehicle management, community interaction, service location, education content delivery, AI assistance, and appointment management. These subsystems communicate through well-defined RESTful APIs to ensure loose coupling and scalability.

The primary design goals of the system are:

- Modularity and separation of concerns
- Scalability to support growing user bases
- Maintainability through clean architecture principles
- Reliability and data consistency
- Ease of integration with future external services

This document focuses on describing how these goals are achieved through architectural decisions, subsystem decomposition, and detailed component design.

1.3 Development Methods & Contingencies

Development Methodology

The system design follows an object-oriented and component-based design approach, supported by Unified Modeling Language (UML) diagrams to model system structure and behavior. The design process aligns with an Agile and Iterative development methodology, allowing incremental refinement of system components based on continuous validation and feedback.

The following design practices are applied:

- Layered Architecture to separate presentation, business logic, and data access layers
- Modular Design to isolate system functionalities into independent subsystems
- UML Modeling including use case diagrams, class diagrams, sequence diagrams, and deployment diagrams
  RESTful API Design for communication between mobile clients and the backend server

Flutter is used for cross-platform mobile development, ensuring consistency across Android and iOS devices. The backend design leverages Node.js for scalability and asynchronous processing, while the database design ensures data integrity and efficient query handling.

Design Contingencies and Risk Handling

Several contingencies may affect system design and implementation:

- **External API Limitations:** Dependency on Google Maps API and AI services may be affected by usage limits or service changes. As a contingency, API usage will be optimized, and fallback behaviors (e.g., limited functionality notifications) will be implemented.
- **Network Availability:** Since core system features require internet connectivity, temporary network failures may disrupt functionality. The design includes graceful error handling and user feedback mechanisms.
- **Scalability Constraints:** Initial deployment may face increased load as user adoption grows. The backend is designed to be horizontally scalable using containerization (Docker) and cloud deployment strategies.
- **Requirement Evolution:** Minor requirement changes may arise during development. The modular design allows individual components to be modified without impacting the entire system.

These contingencies are addressed through flexible architecture, clear interface contracts, and adherence to industry-standard design principles.

2. System Architecture

2.1 Subsystem decomposition

2.1.1 Level 1: High- level architecture

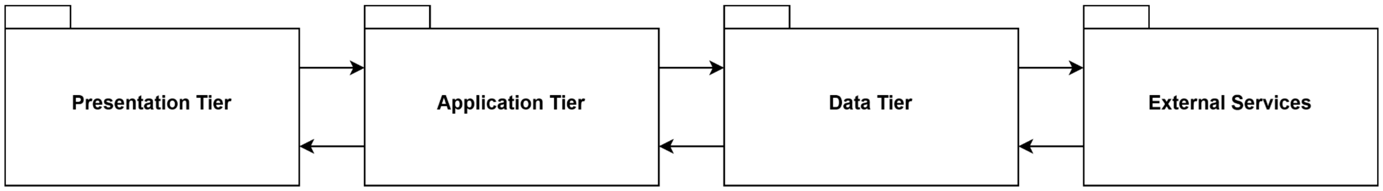

Figure 1: High- level architecture

2.1.2 Level 2: Application Subsystems

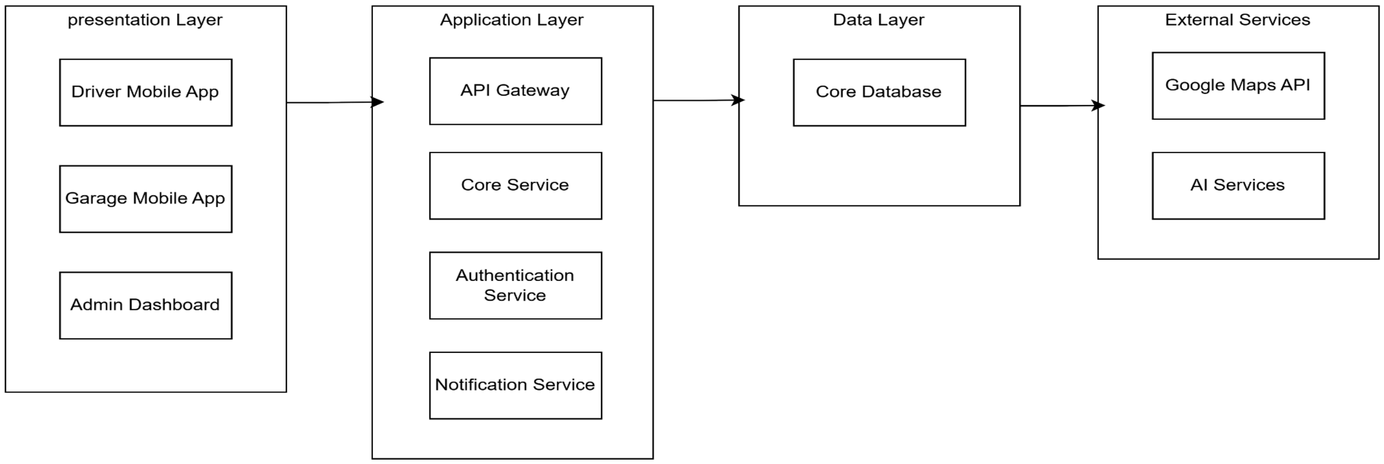

Figure 2: Application Subsystems

2.1.3 Level 3: Core Service Decomposition

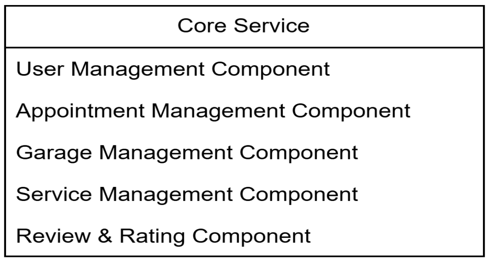
Core Service
User Management Component
Appointment Management Component
Garage Management Component
Service Management Component
Review & Rating Component

Figure 3: Core Service Decomposition

2.2 Hardware/software mapping

UML Deployment diagram

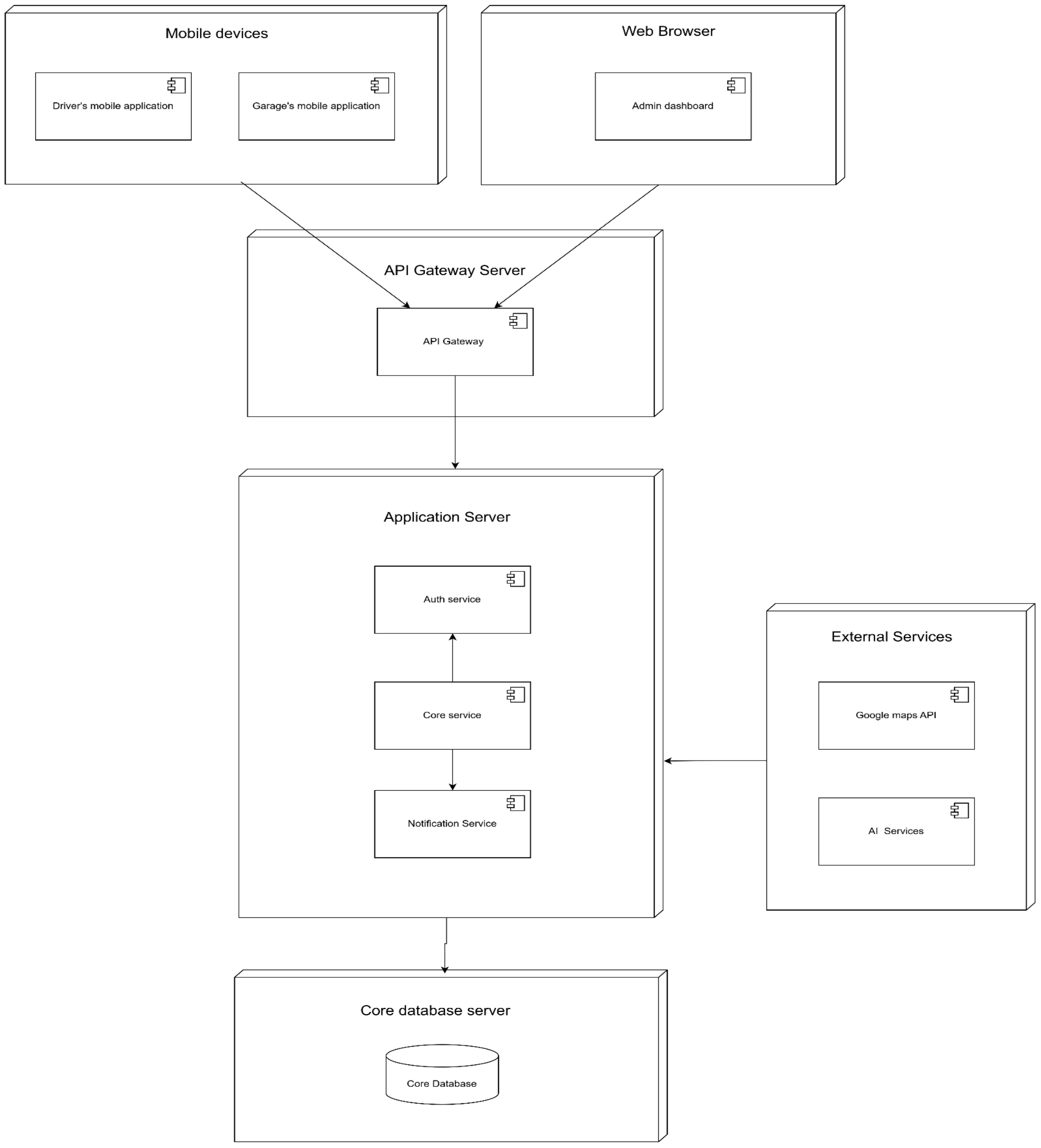

Figure 4: Hardware/software mapping

3. Object Model

3.1 Class Diagram

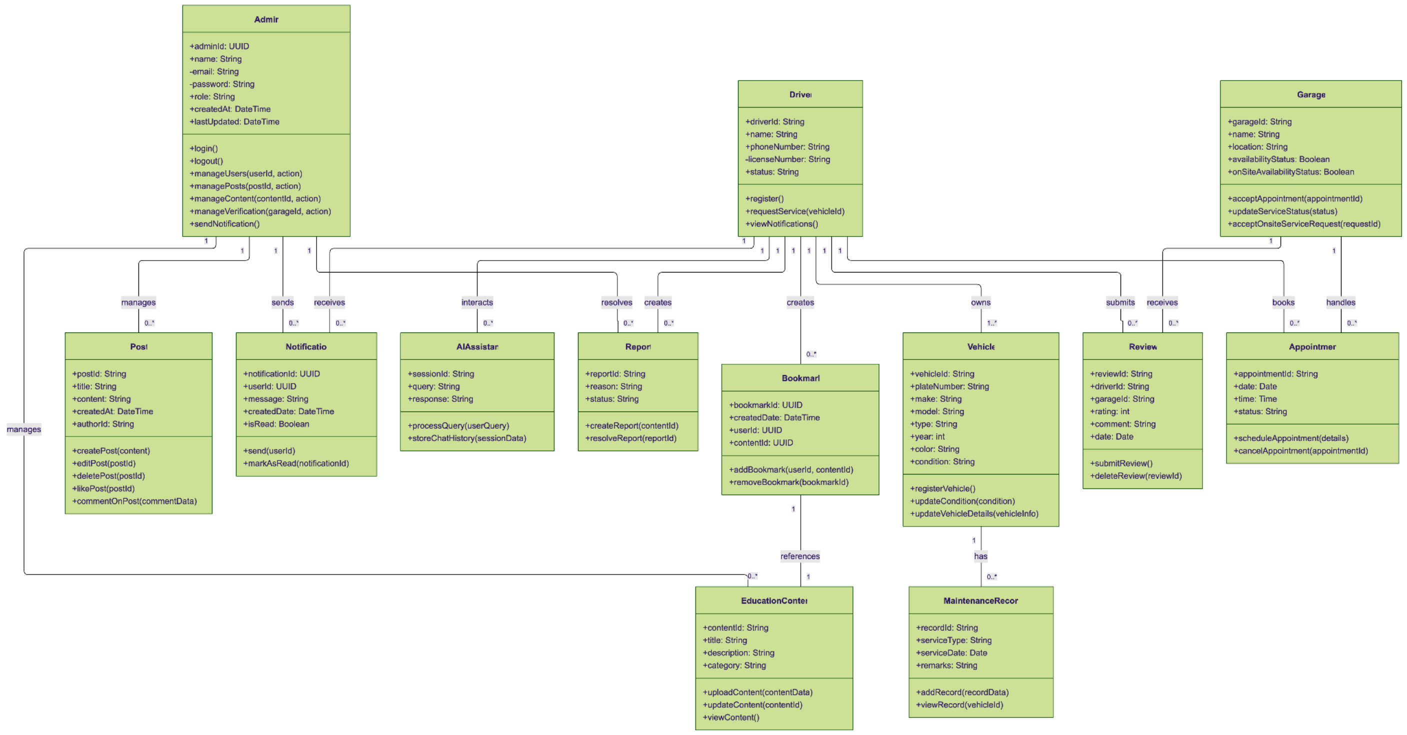

Figure 5: Hardware/software mapping

3.2 Use Case Diagram

3.2.1 Auth and User Management

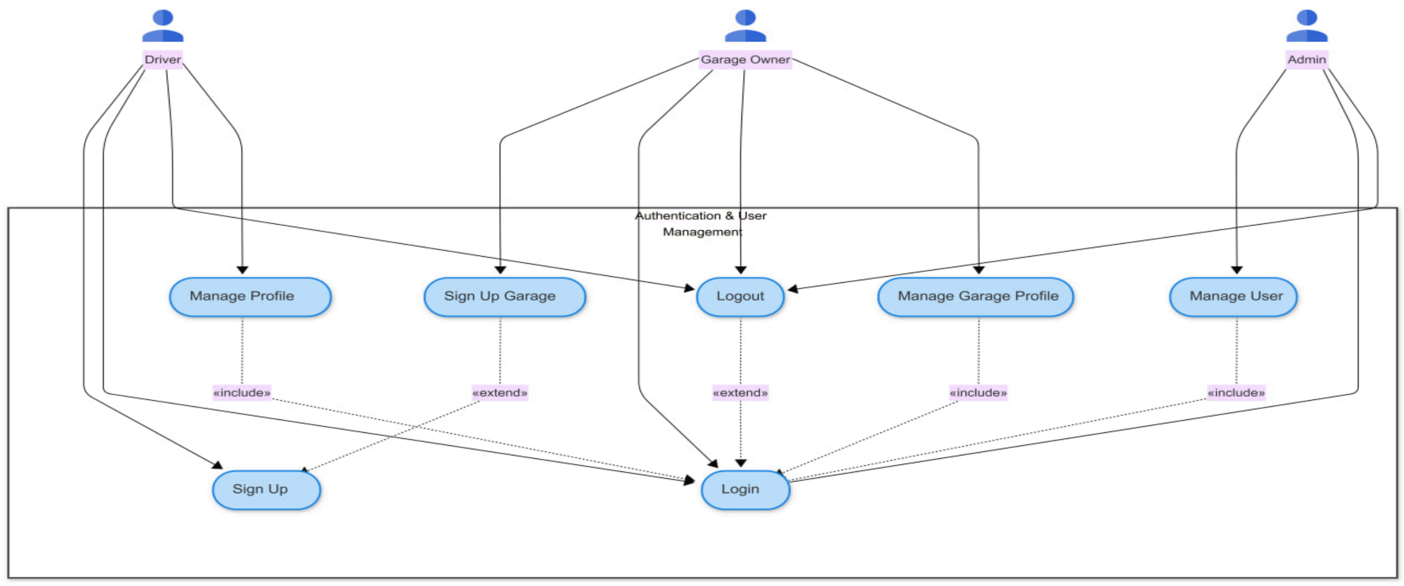

Figure 6: Auth and User Management Use Case Diagram

3.2.2 Vehicle Maintenance & AI Assistance

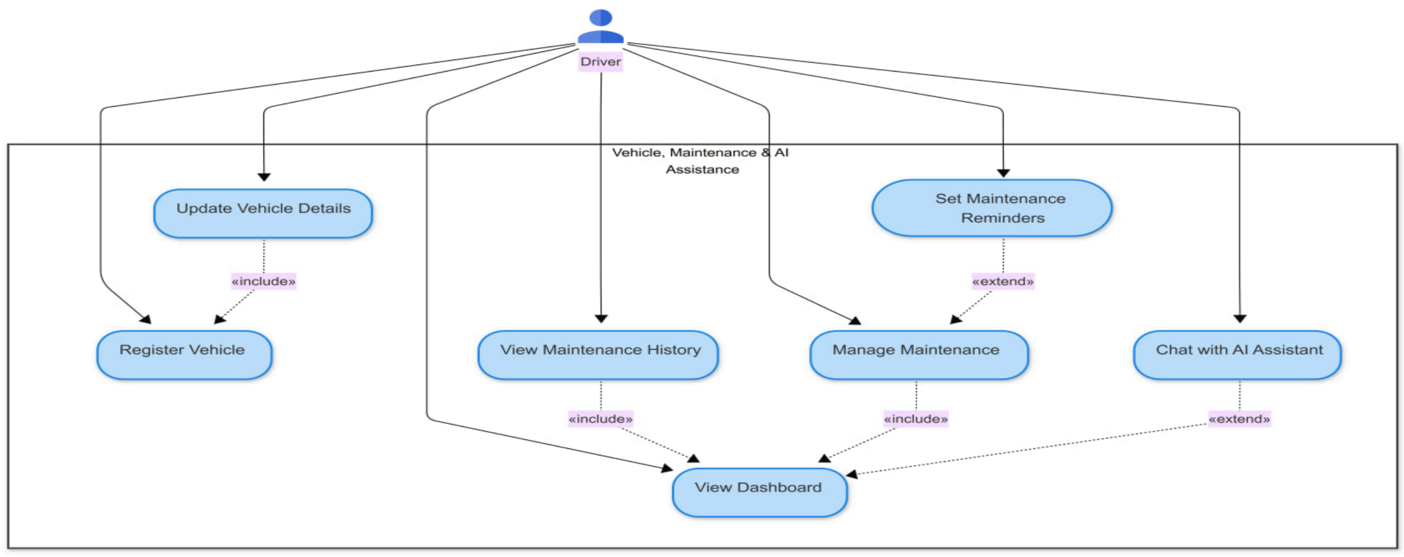

Figure 7: Vehicle Maintenance & AI Assistance Use Case Diagram

3.2.3 Garage Service & Appointment

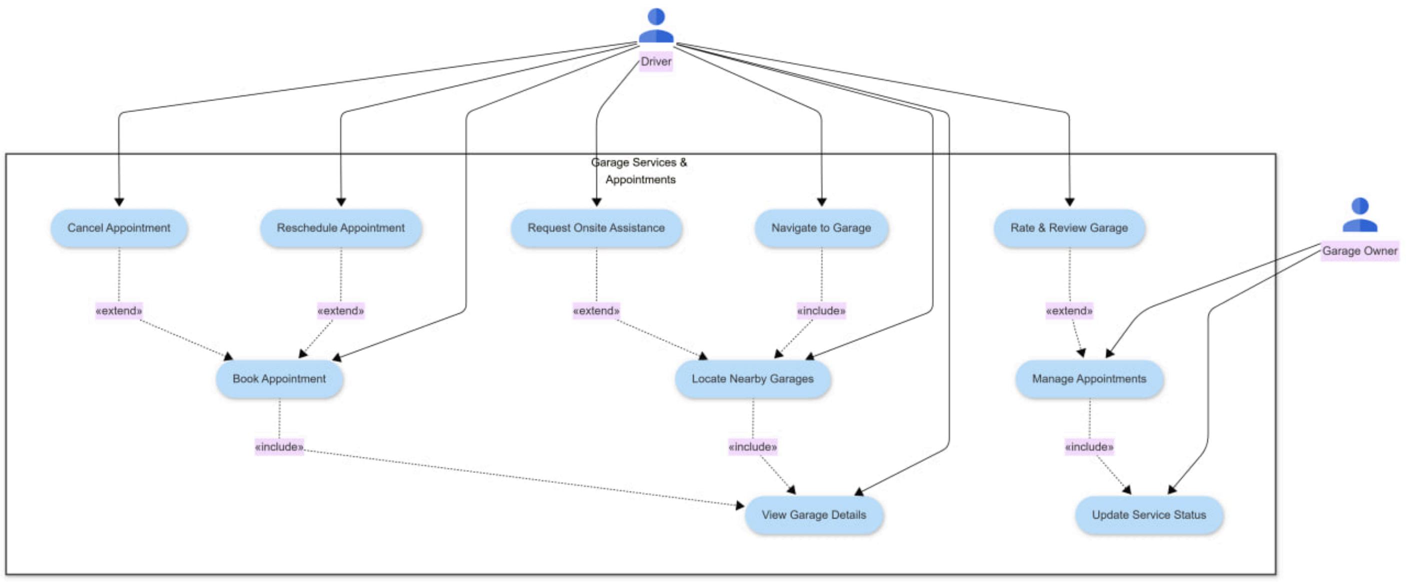

Figure 8: Garage Service & Appointment Use Case Diagram

3.2.4 Community & Learning Center

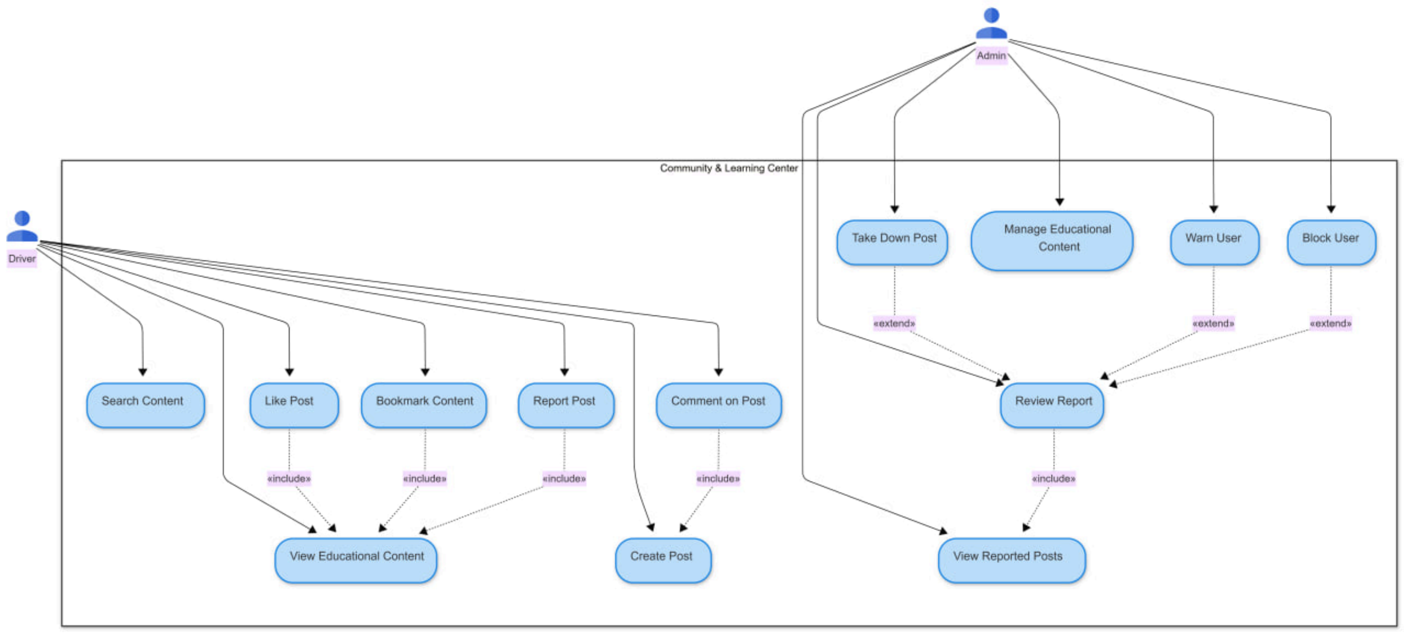

Figure 9: Community & Learning Center Use Case Diagram

3.3 Sequence Diagram

Show how processes operate with one another and in what order. Depict the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

Sequence Diagram: Driver Registration

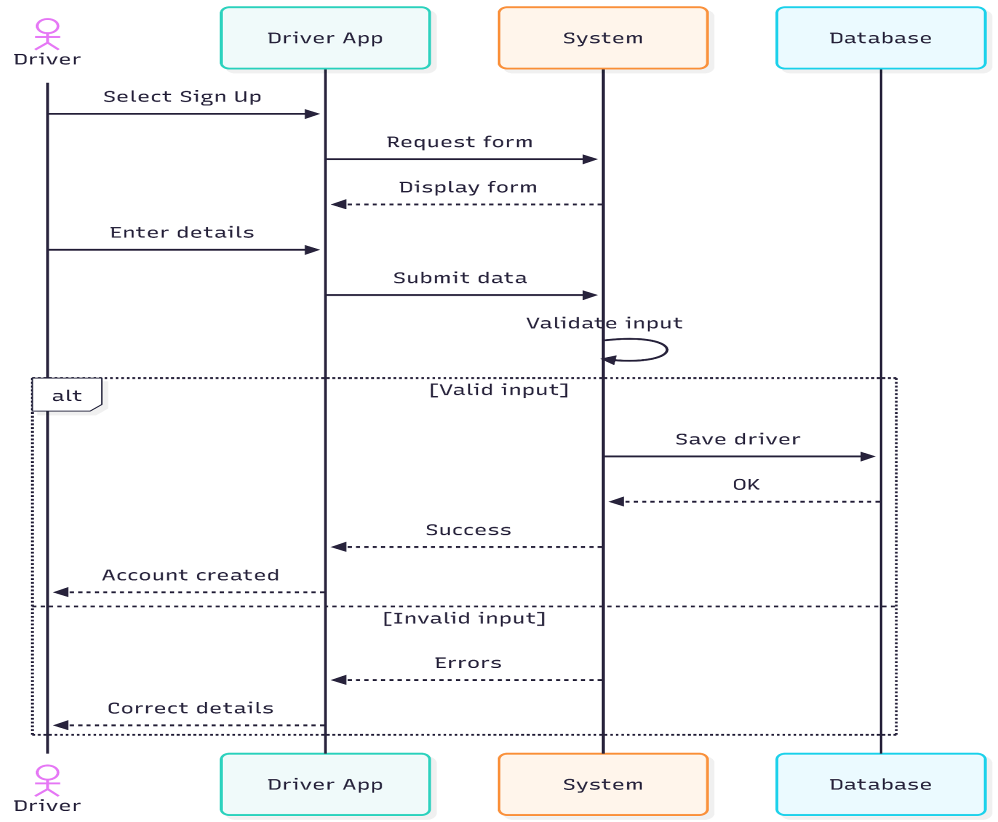

Figure 10: Driver Registration Sequence Diagram

Sequence Diagram: Garage registration

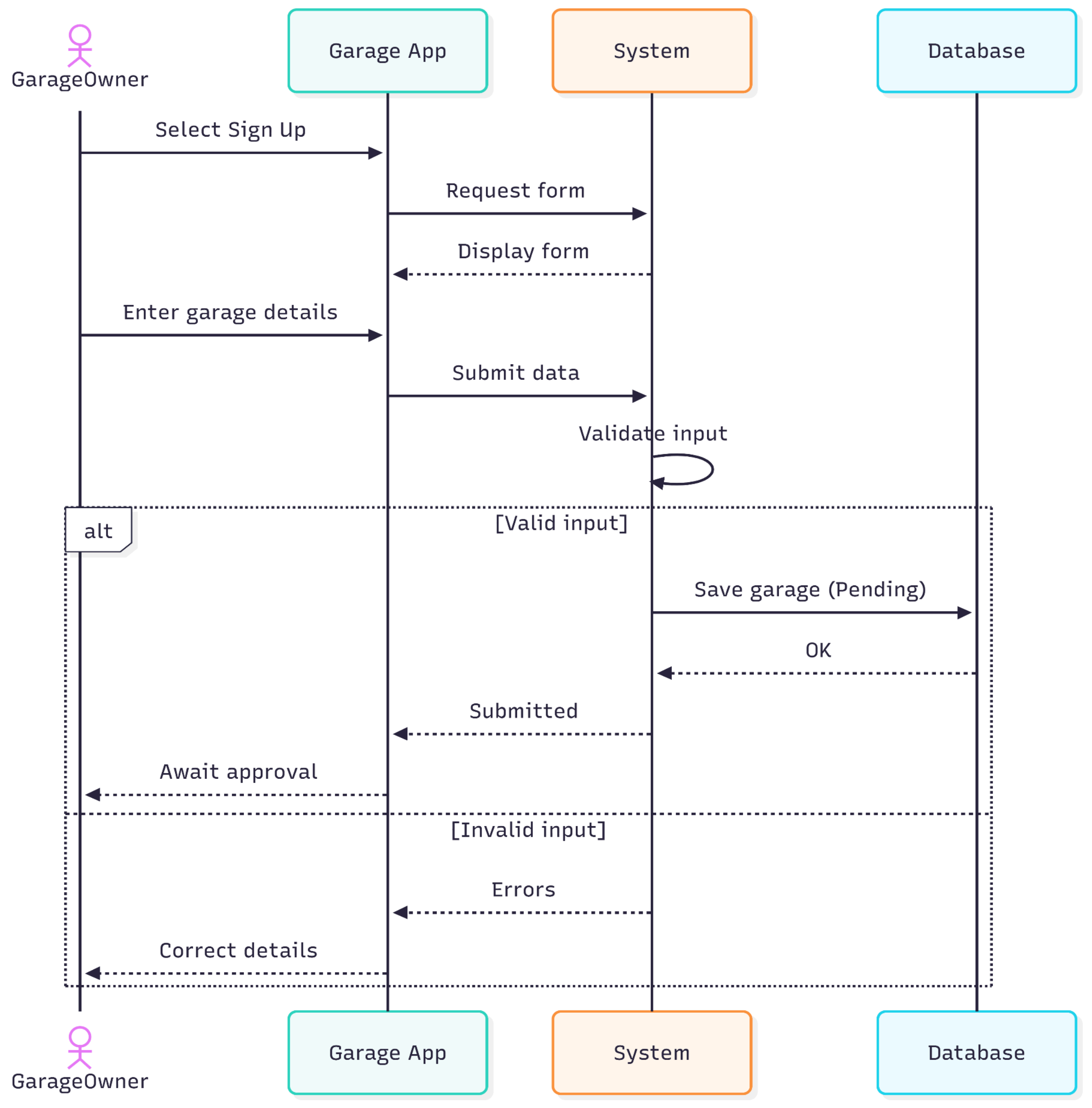

Figure 11: Garage registration Sequence Diagram

Sequence Diagram: Authentication

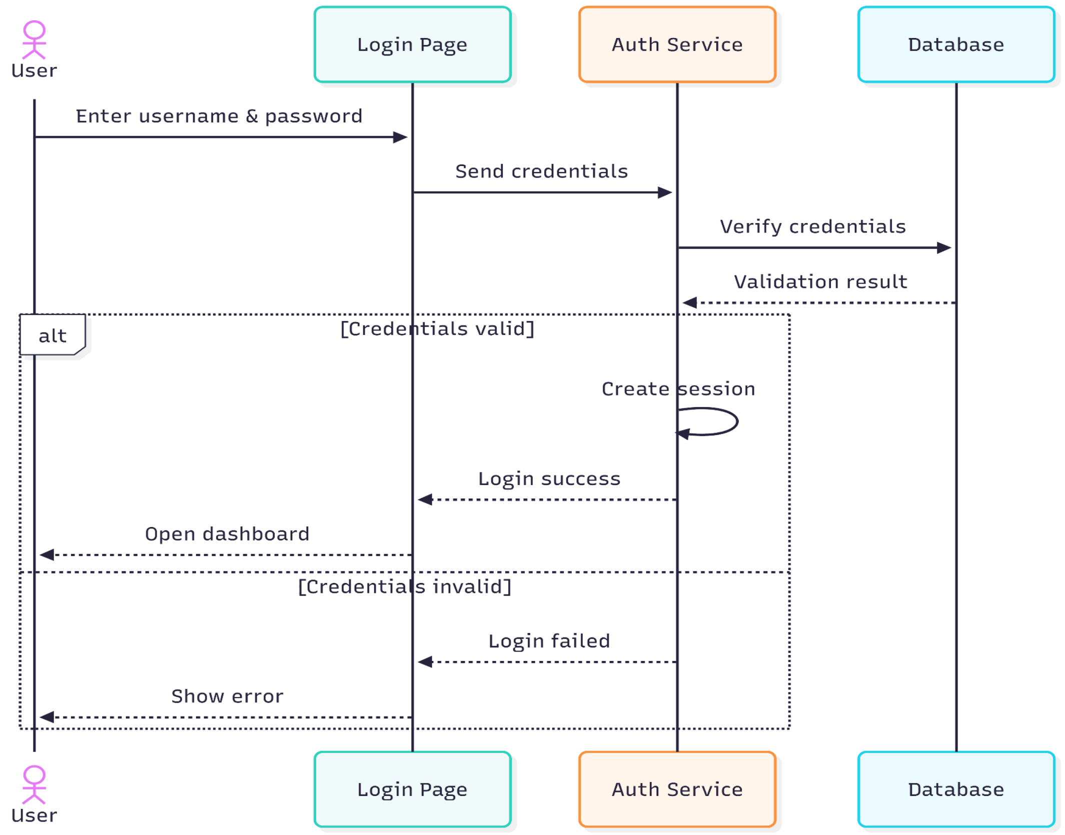

Figure 12: Authentication Sequence Diagram

Sequence Diagram: Community Post

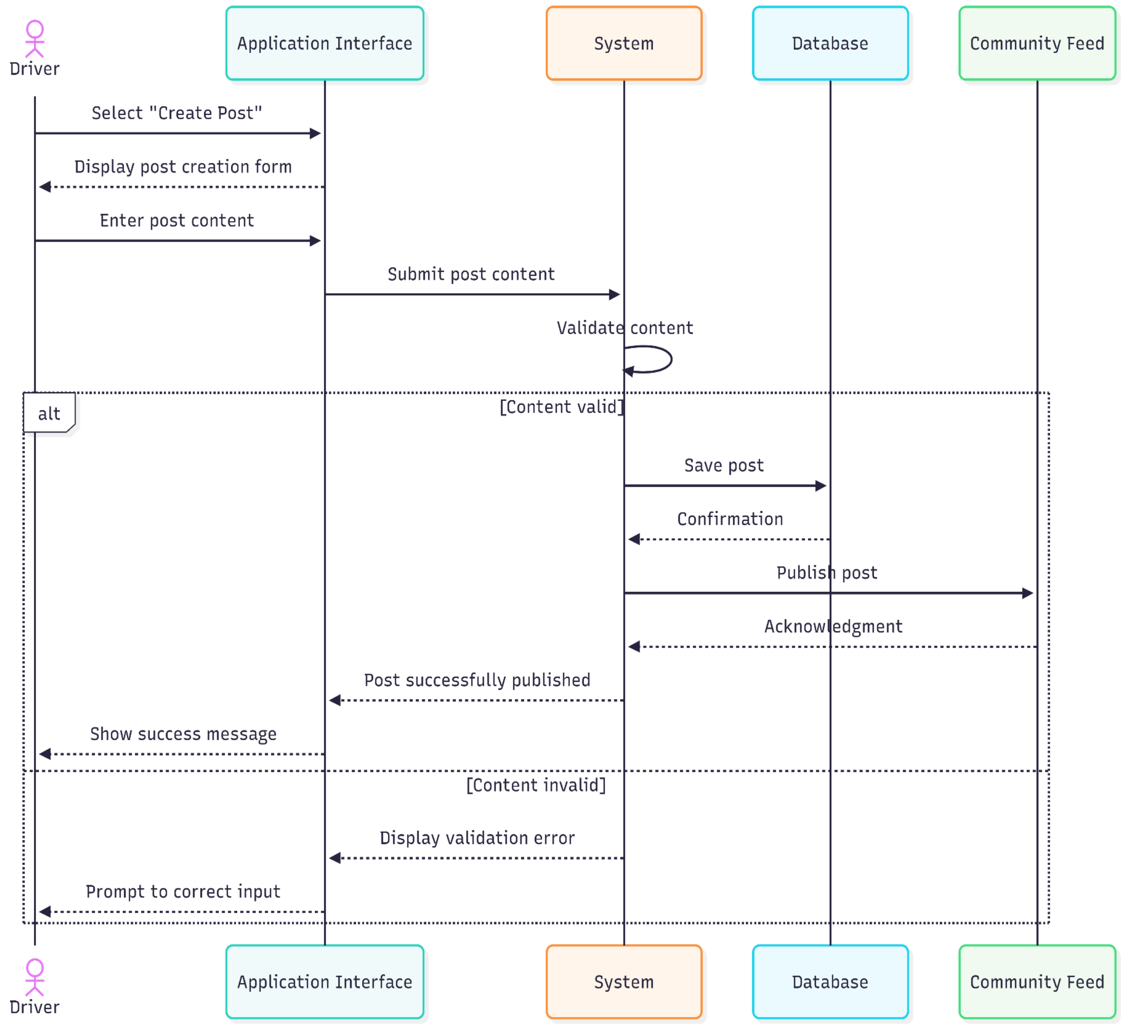

Figure 13: Community Post Sequence Diagram

Sequence Diagram: Education content

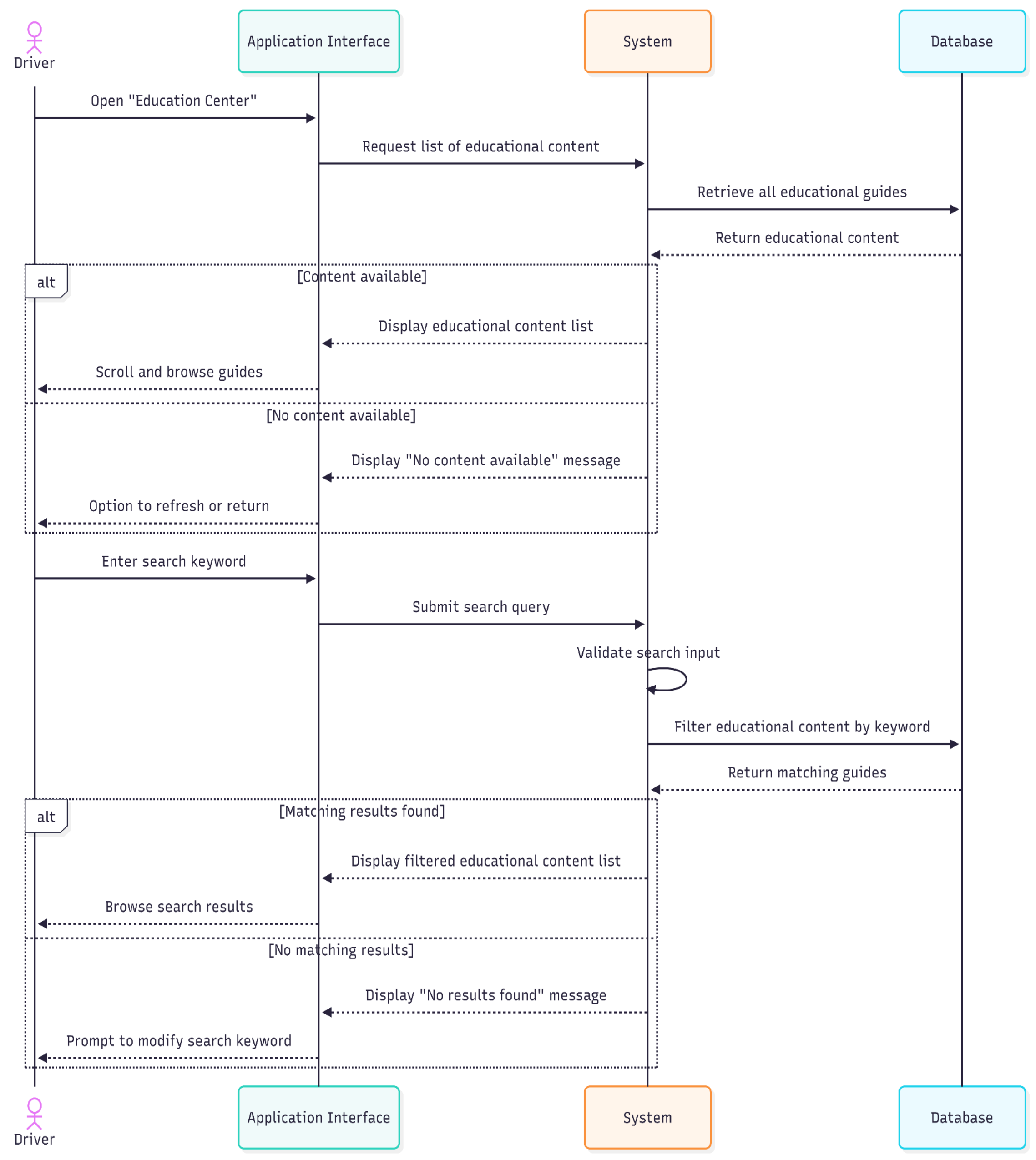
Driver
Application Interface
System
Database
Open "Education Center"
Request list of educational content
Retrieve all educational guides
Return educational content
alt
[Content available]
Display educational content list
Scroll and browse guides
[No content available]
Display "No content available" message
Option to refresh or return
Enter search keyword
Submit search query
Validate search input
Filter educational content by keyword
Return matching guides
alt
[Matching results found]
Display filtered educational content list
Browse search results
[No matching results]
Display "No results found" message
Prompt to modify search keyword

Figure 14: Education content Sequence Diagram

Sequence Diagram: Vehicle Registration

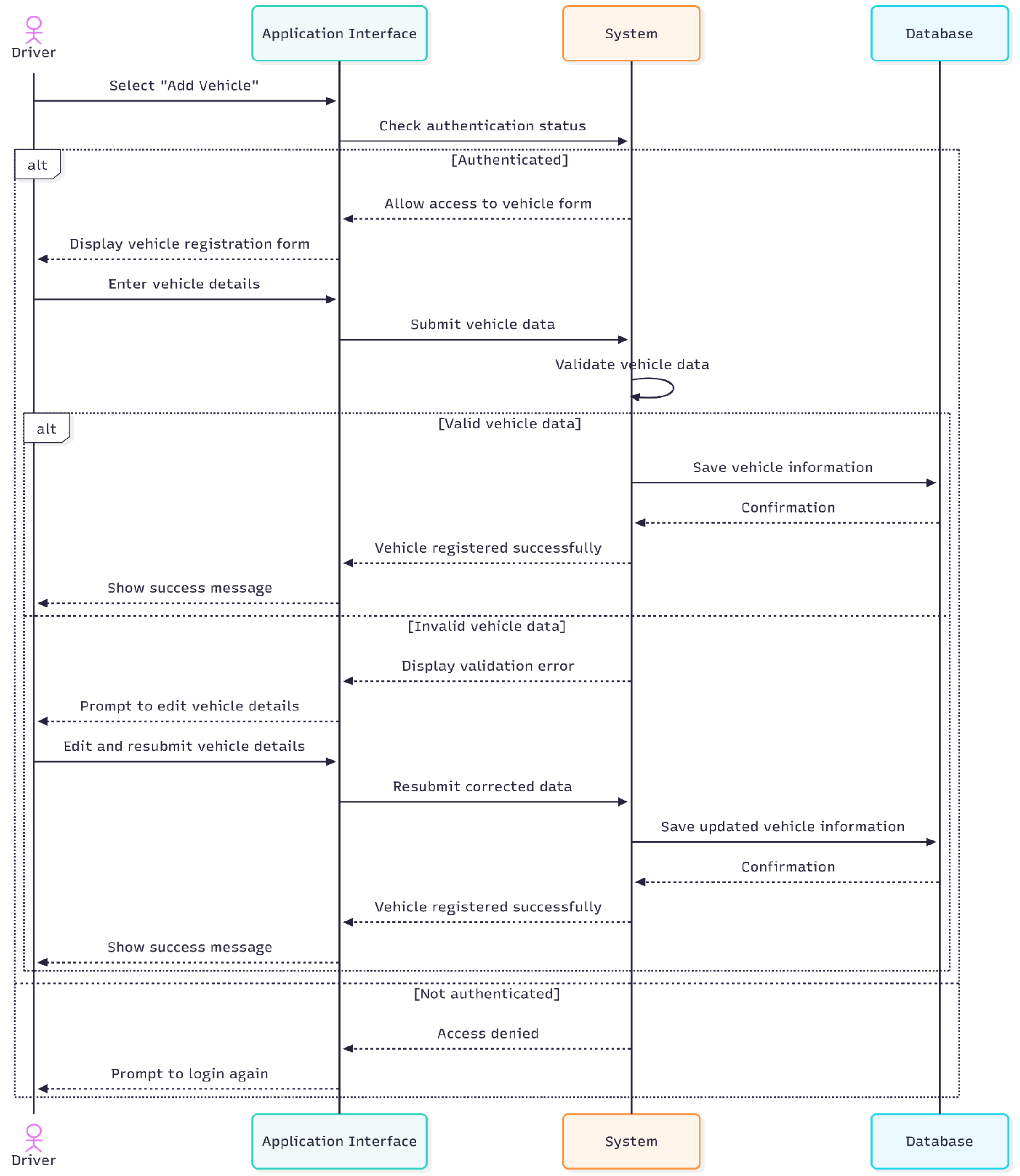

Figure 15: Vehicle Registration Sequence Diagram

Sequence Diagram: Book Garage Appointment

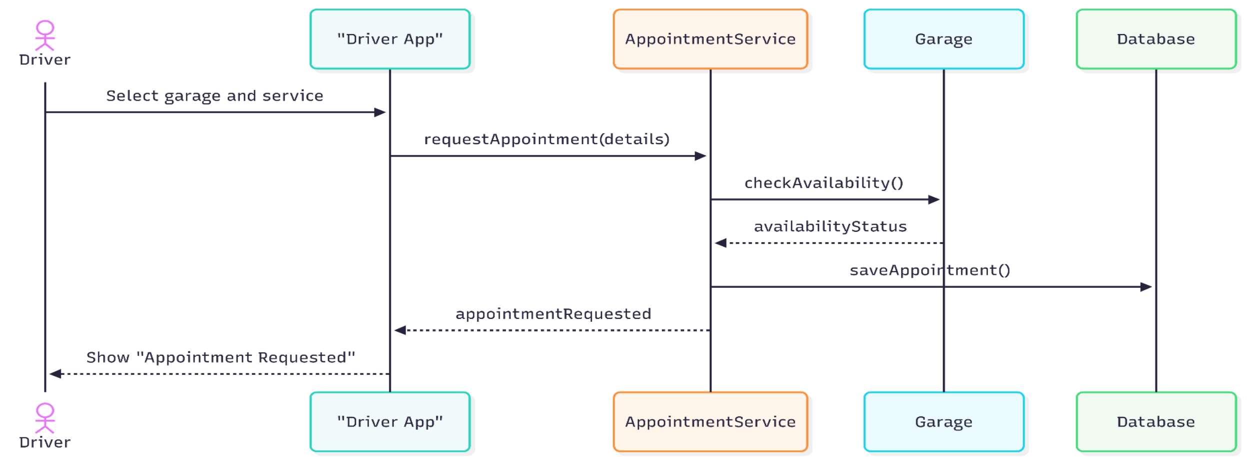

Figure 16: Book Garage Appointment Sequence Diagram

Sequence Diagram: Appointment Approval & Notification

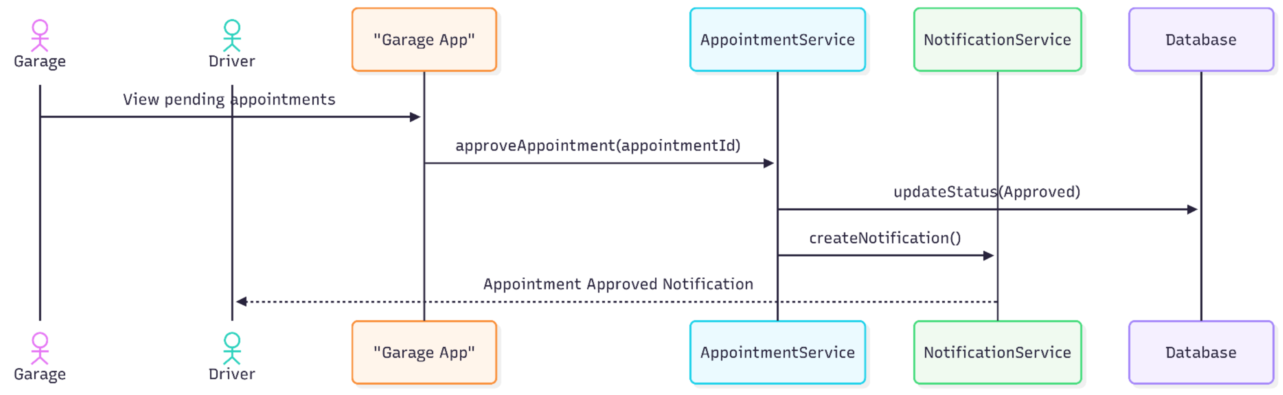

Figure 17: Appointment Approval & Notification Sequence Diagram

Sequence Diagram: Maintenance Service Completion

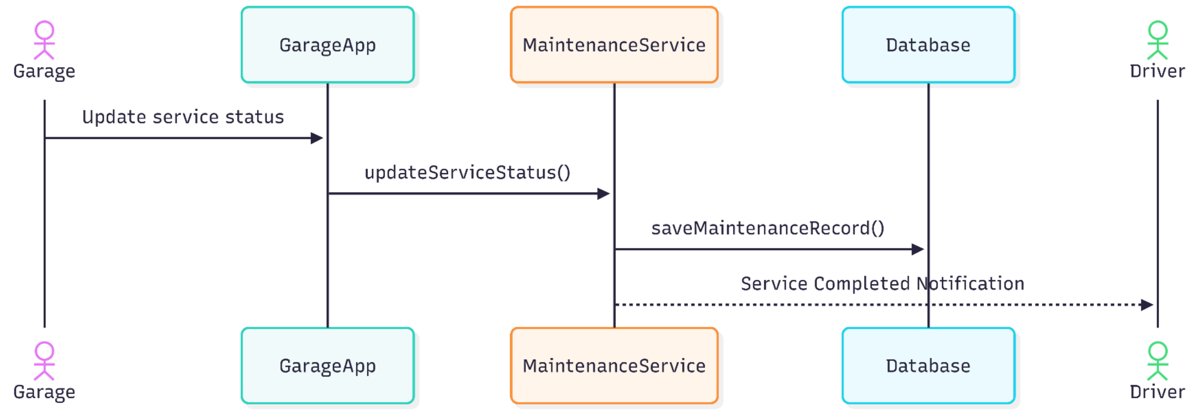

Figure 18: Maintenance Service Completion Sequence Diagram

Sequence Diagram: View Maintenance History

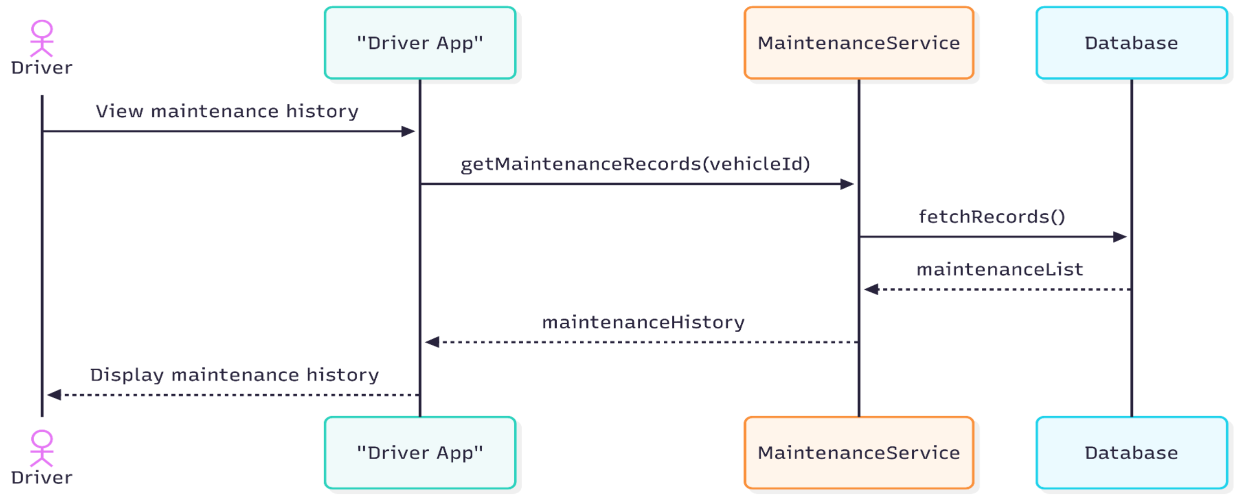

Figure 19: View Maintenance History Sequence Diagram

Sequence Diagram: AI Assistant

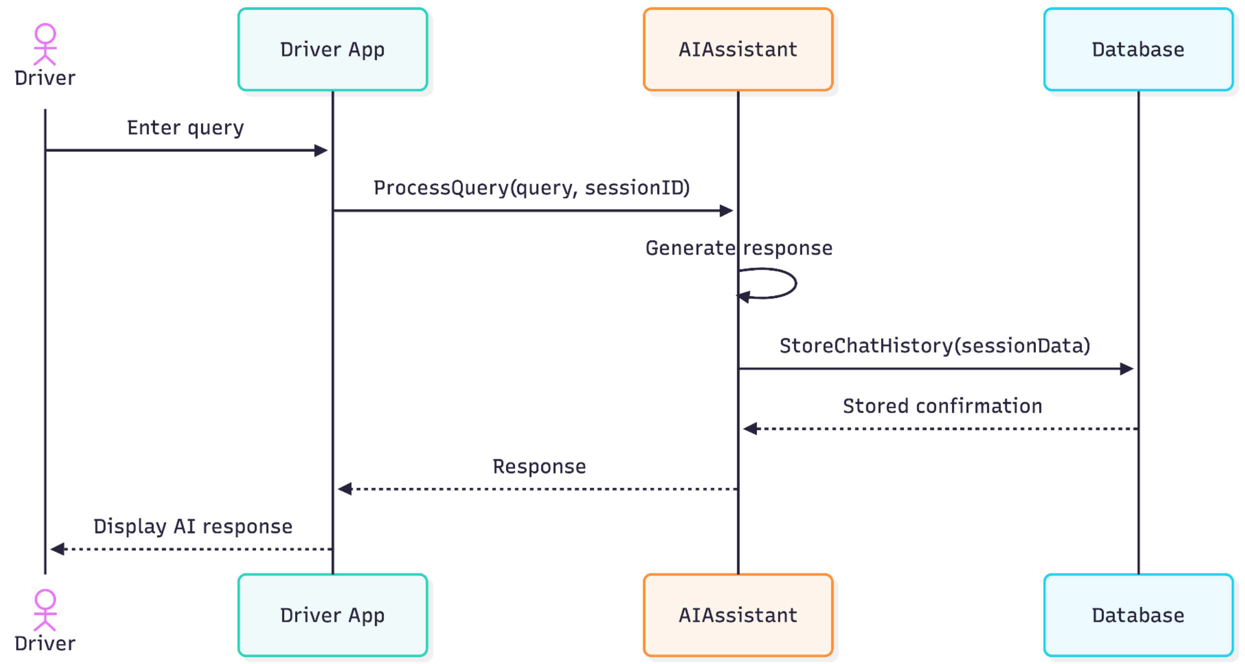

Figure 20 : AI Assistant Sequence Diagram

Sequence Diagram: Feedback

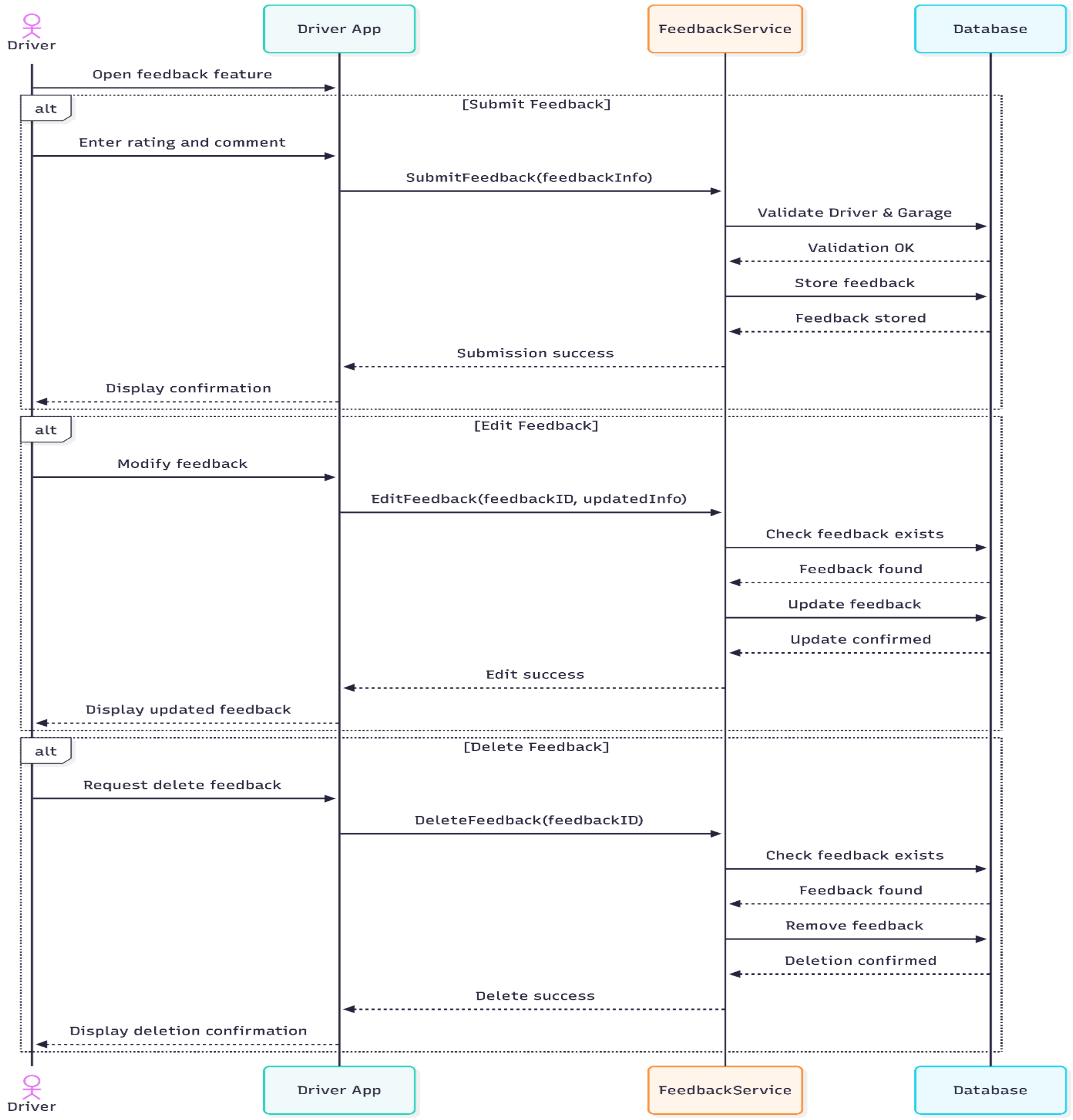

Figure 21: Feedback Sequence Diagram

4. Detailed Design

This section presents the detailed design of the classes identified in the class diagram. Additional methods are included from the sequence and state chart diagrams. Each class is described using class structure, attribute description, and operation description.

Table: 1 Admin Class

Admin

+admin_id : UUID
+name : String
-email : String
-password : String
+role : String
+created_at: datetime
+last_updated: datetime

+Login()
+Logout()
+ManageUsers(userId, action)
+ManageContent(contentId, action)
+ManageVerification(garageId, action)
+ManagePosts(postId, action)
+SendNotification()

Table 1.1: Attribute Description for Admin Class

Attribute	Type	Visibility	Invariant
admin_id	UUID	Public	Must be unique and not NULL
name	String	Public	Must not be NULL; must contain only alphabetic characters
email	String	Private	Must contain @ and .; Must be unique
password	String	Private	Must be at least 8 characters

role	String	Public	Must not be NULL;
created_at	datetime	public	Timestamp when admin was created
last_updated	datetime	public	Timestamp when the admin was last updated

Table: 1.2 Operation Description for Admin Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
Login	Public	Boolean	Email, Password	Admin account exists	Admin authenticated
ManageUsers	Public	void	userId, action	User exists	User updated (blocked/unblocked/, warned etc.)
ManagePosts	Public	void	postId, action	Post exists	Post approved or deleted
Manage Content	Public	void	contentId, action	Content exists	Content added, updated, or deleted
Manage Verifications	Public	void	garageId, action	Garage exists	Garage approved or rejected
Send Notification	Public	void	Message	Message exists	Notification sent

Table: 2 Driver Class

Driver

+DriverID: String
+Name: String

+PhoneNumber: String
-LicenseNumber: String
+Status: String

+Register()
+RequestService()
+ViewNotifications()

Table 2.1: Attribute Description for Driver Class

Attribute	Type	Visibility	Invariant
DriverID	String	Public	Must be unique
Name	String	Public	Name ≠ NULL
PhoneNumber	String	Public	Must start with +251 or 09
LicenseNumber	String	Private	Must be valid
Status	String	Public	Must be either Active or Inactive

Table 2.2: Operation Description for Driver Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
Register	Public	void	DriverInfo	Driver is not registered	Driver is registered in the system
RequestService	Public	void	VehicleID	Driver exists in the system	Service requested is created
View Notifications	Public	List	-	Notifications exist for the driver	Notifications are displayed to the driver

Table: 3 Garage Class

Garage

+GarageID: String
+Name: String
+Location: String
+AvailabilityStatus: Boolean

+OnSiteAvailabilityStatus: Boolean

+AcceptAppointment()
+UpdateServiceStatus()

+AcceptOnsiteServiceRequest()

Table 3.1: Attribute Description for Garage Class

Attribute	Type	Visibility	Invariant
GarageID	String	Public	Unique
Name	String	Public	Not NULL
Location	String	Public	Valid address
AvailabilityStatus	Boolean	Public	True or False
OnSiteAvailability Status	Boolean	Public	True or False

Table 3.2: Operation Description for Garage Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
AcceptAppointment	Public	void	AppointmentID	Appointment exists	Appointment accepted
UpdateServiceStatus	Public	void	Status	Service ongoing	Status updated

AcceptOnsiteServiceRequest	Public	void	RequestId	OnSiteAvailabilityStatus	Request accepted

Table: 4 Appointment Class

Appointment

+AppointmentID : String
+Date : Date
+Time : Time
+Status : String

+ScheduleAppointment()
+CancelAppointment()

Table: 4.1 Attributes Description for Appointment Class

Attribute	Type	Visibility	Invariant
AppointmentID	String	Public	Unique
Date	Date	Public	≥ current date
Time	Time	Public	Valid
Status	String	Public	Scheduled/Cancelled

Table: 4.2 Operation Description for Appointment Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
ScheduleAppointment	Public	void	Details	Slot available	Appointment scheduled

CancelAppointment	Public	void	AppointmentID	Exists	Appointment cancelled

Table: 5 Notification Class

Notification

+notification_id : UUID
+user_id: UUID
+message : string
+created_date : datetime
+is_read : boolean

+Send()
+MarkAsRead()

Table: 5.1 Attributes Description for Notification Class

Attribute	Type	Visibility	Invariant
notificationID	UUID	Public	Unique and not NULL
user_id	UUID	public	Unique and not null
Message	String	Public	Not NULL
created_date	datetime	Public	Valid
is_read	boolean	Public	True or False

Table: 5.2 Operation Description for Notification Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition

Send	Public	void	UserID	User exists	Notification created and sent
MarkAsRead	Public	void	NotificationID	Notification exists	Notification marked read

Table: 6 Vehicle Class

Vehicle

+ vehicleId : String

+ plateNumber : String

+ make : String

+ model : String

+ type : String

+ year : int

+ color : String

+ condition : String

+RegisterVehicle()
+UpdateCondition()

+ updateVehicleDetails()

Table: 6.1 Attributes Description for Vehicle Class

Attribute	Type	Visibility	Invariant
VehicleID	String	Public	Unique
PlateNumber	String	Public	Valid format
make	String	Public	Not NUL
Model	String	Public	Not NULL
Type	String	Public	Not NUL
Year	Int	Public	> 1990
Color	String	Public	Not NUL
Condition	String	Public	Good/Fair/Ba d

Table: 6.2 Operation Description for Vehicle Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
RegisterVehicle	Public	void	VehicleInfo	Vehicle Not registered	Vehicle data Registered and stored successfully
UpdateCondition	Public	void	Condition	Vehicle Exists	Vehicle Updated
updateVehicle Details	Public	void	VehicleInfo	Vehicle Exists	Vehicle details updated

Table: 7 Post Class

Post

+ postId : String

+ title : String

+ content : String

+ createdAt : DateTime

+ authorId : String

+CreatePost()
+EditPost()
+DeletePost()

+likePost**()**

+commentOnpost**()**

Table: 7.1 Attributes Description for Post Class

Attribute	Type	Visibility	Invariant
PostID	String	Public	Unique
Title	String	Public	Not NULL
Content	String	Public	Not NULL
createdAt	DateTime	Public	System generated
authorId	String	Public	Valid user identifier

Table: 7.2 Operation Description for Post Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
CreatePost	Public	void	Content	Authenticated user	Post Created

EditPost	Public	void	PostID	Post Exists& owned	Post content updated
DeletePost	Public	void	PostID	Post Exists& owned	Post removed from system
likePost	Public	Void	PostID	User authenticated	Like count incremented
commentOnPost	Public	Void	CommentData	User authenticated	Comment added to the post

Table: 8 Education Content Class

EducationContent

+ContentID : String
+Title : String
+Description : String
+Category : String

+UploadContent()
+UpdateContent()
+ViewContent()

Table: 8.1 Attributes Description for Education Content Class

Attribute	Type	Visibility	Invariant
ContentID	String	Public	Unique
Title	String	Public	Not NULL
Description	String	Public	Not NULL
Category	String	Public	Not NULL

Table: 8.2 Operation Description for Education Content Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
UploadContent	Public	void	ContentData	Admin exists	Uploaded
UpdateContent	Public	void	ContentID	Exists	Updated
ViewContent	Public	List	-	Exists	Displayed

Table: 9 Maintenance Record Class

MaintenanceRecord

+RecordID : String
+ServiceType : String
+ServiceDate : Date
+Remarks : String

+AddRecord()
+ViewRecord()

Table: 9.1 Attributes Description for Maintenance Record Class

Attribute	Type	Visibility	Invariant
RecordID	String	Public	Unique
ServiceType	String	Public	Not NULL
ServiceDate	Date	Public	≤ current date
Remarks	String	Public	Optional

Table: 9.2 Operation Description for Maintenance Record Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
AddRecord	Public	void	RecordData	Service completed	Stored
ViewRecord	Public	List	VehicleID	Exists	Displayed

Table: 10 Review Class

Review Class

+ ReviewID : String
+ DriverID : String
+ GarageID : String
+ Rating : Integer
+ Comment : String
+ Date : Date

+ SubmitReview()
+ DeleteReview()

Table 10.1: Attributes Description for Review Class

Attribute	Type	Visibility	Invariant
ReviewID	String	Public	Must be unique
DriverID	String	Public	Must reference a valid Driver
GarageID	String	Public	Must reference a valid Garage
Rating	Integer	Public	Must be between 1 and 5
Comment	String	Public	May be empty
Date	Date	Public	Must be valid date

Table: 11 Report Class

Report

+ReportID : String
+Reason : String
+Status : String

+CreateReport()
+ResolveReport()

Table: 11.1 Attributes Description for Report Class

Attribute	Type	Visibility	Invariant
ReportID	String	Public	Unique
Reason	String	Public	Not NULL
Status	String	Public	Pending/Resolved

Table: 11.2 Operation Description for Report Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
CreateReport	Public	void	ContentID	Exists	Created
ResolveReport	Public	void	ReportID	Admin exists	Resolved

Table: 12 Bookmark Class

Bookmark

+BookmarkID : UUID
+CreatedDate : datetime
+user_id: UUID
+content_id: UUID

+AddBookmark()
+RemoveBookmark()

Table: 12.1 Attributes Description for Bookmark Class

Attribute	Type	Visibility	Invariant
bookmark_id	UUID	Public	Must be unique and not NULL
created_date	datetime	Public	Must be a valid timestamp
user_id	UUID	public	Must reference a valid existing user
content_id	UUID	public	Must reference valid existing content
BookmarkID	UUID	Public	Must be unique and not NULL

Table: 12.2 Operation Description for Bookmark Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
AddBookmark	Public	void	user_id, content_id	User and content exist	Bookmark created and stored
RemoveBookmark	Public	void	bookmark_id	Bookmark exists	Bookmark removed

Table: 13 AI Assistant Class

AI Assistant

+SessionID : String
+Query : String
+Response : String

+ProcessQuery()
+StoreChatHistory()

Table: 13.1 Attributes Description for AI Assistant Class

Attribute	Type	Visibility	Invariant
SessionID	String	Public	Unique
Query	String	Public	Not NULL
Response	String	Public	Generated

Table: 13.2 Operation Description for AI Assistant Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
ProcessQuery	Public	String	UserQuery	Authenticated	Response generated
StoreChatHistory	Public	void	SessionData	Exists	Stored

Table 14.2: Operation Description for Feedback Class

Operation	Visibility	Return Type	Arguments	Pre-Condition	Post-Condition
SubmitFeedback	Public	void	FeedbackInfo	Driver and Garage exist	Feedback is stored in the system
EditFeedback	Public	void	FeedbackID, UpdatedInfo	Feedback exists	Feedback is updated

DeleteFeedback	Public	void	FeedbackID	Feedback exists	Feedback is removed from the system

Reference

Web resource:

https://www.tutorialspoint.com/uml/uml_class_diagram.htm

Software design description. Wikipedia. (Accessed: Jan 3, 2026)
https://en.wikipedia.org/wiki/Software_design_description